# CHAPTER – FOUR

# DESIGN

## 4.1 Overview

This section includes the proposed design diagrams for the loan predictor models. The various design diagrams elaborates the various aspects of the internal structure of the project.

## 4.2 Class Diagram

The class diagram is the main building block of object-oriented modelling. It is used for general conceptual modelling of the systematic of the application, and for detailed modelling translating the models into programming code. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed. Class diagram for the project is as follows:

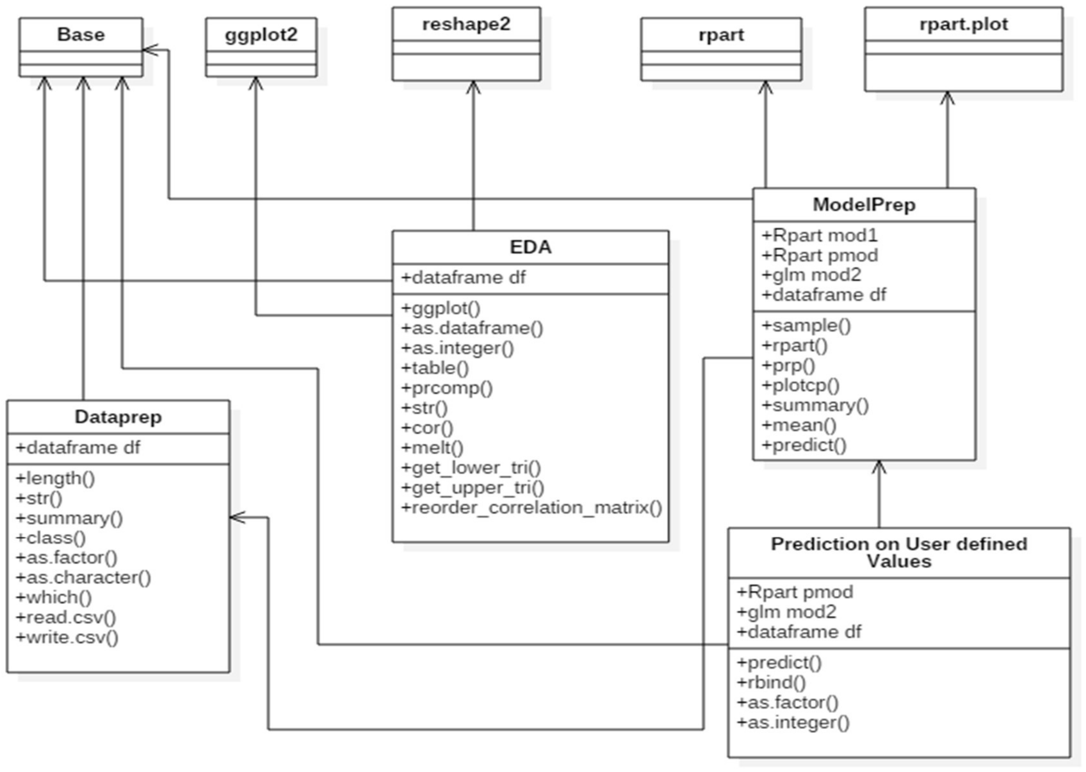

**Fig 4.1** Class Diagram

## 4.3 Object Diagram

An **object diagram** in the Unified Modeling Language (UML), is a diagram that shows a complete or partial view of the structure of a modeled system at a specific time.

In the Unified Modeling Language (UML), an object diagram focuses on some particular set of objects and attributes, and the links between these instances. A correlated set of object diagrams provides insight into how an arbitrary view of a system is expected to evolve over time. In early UML specifications the object diagram is described as:

An object diagram is a graph of instances, including objects and data values. A static object diagram is an instance of a class diagram; it shows a snapshot of the detailed state of a system at a point in time. The use of object diagrams is fairly limited, namely to show examples of data structure.

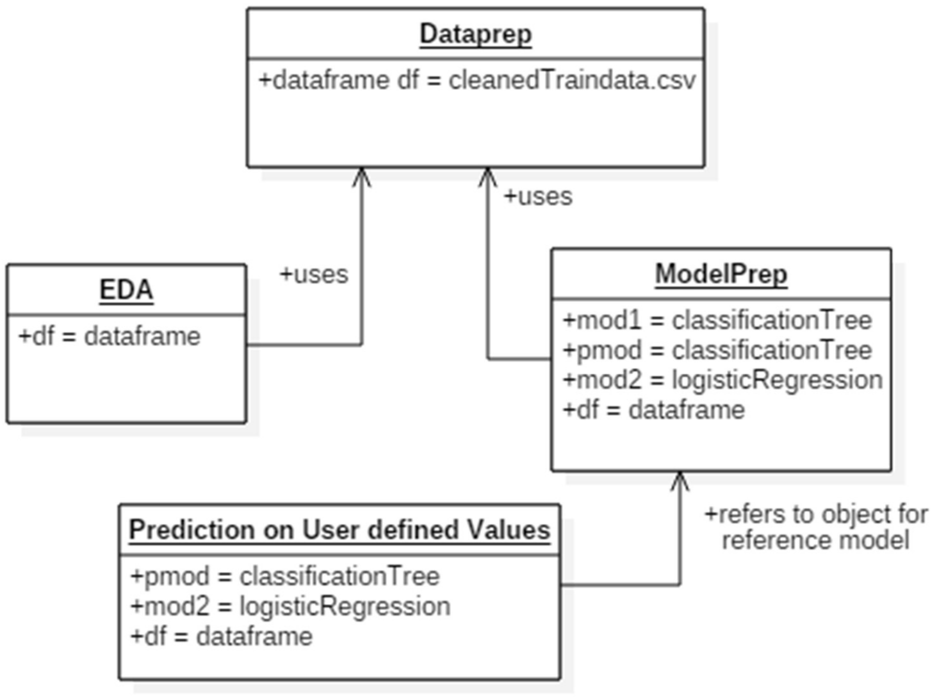
Dataprep
+dataframe df = cleanedTraindata.csv
+uses
+uses
EDA
+df = dataframe
ModelPrep
+mod1 = classificationTree
+pmod = classificationTree
+mod2 = logisticRegression
+df = dataframe
Prediction on User defined Values
+pmod = classificationTree
+mod2 = logisticRegression
+df = dataframe
+refers to object for reference model

**Fig 4.2** Object Diagram

## 4.4 Entity Relationship Diagram

An entity–relationship model (ER model) describes inter-related things of interest in a specific domain of knowledge. An ER model is composed of entity types (which classify the things of interest) and specifies relationships that can exist between instances of those entity types.

In software engineering an ER model is commonly formed to represent things that a business needs to remember in order to perform business processes. Consequently, the ER model becomes an abstract data model that defines a data or information structure that can be implemented in a database, typically a relational database.

ER diagram for the project is as follows:

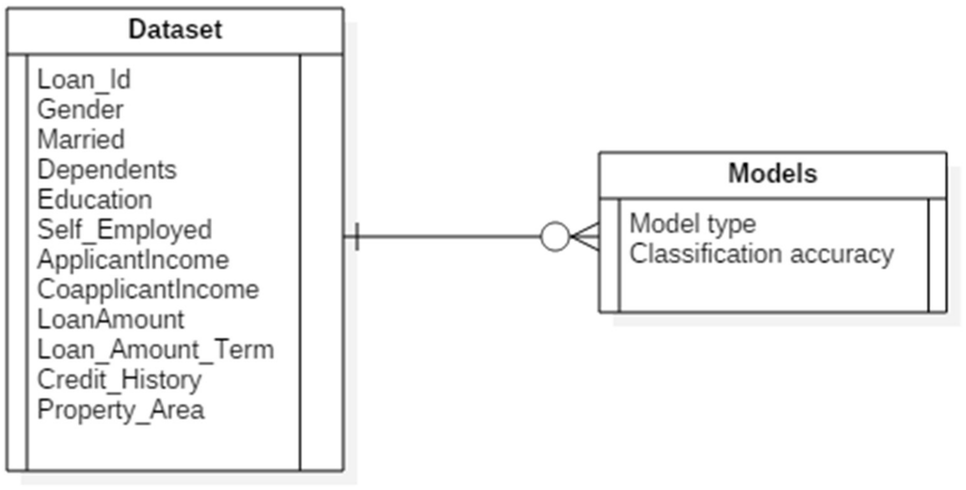

**Fig 4.3** ER Diagram

## 4.5 Use Case Diagram

UML Use Case Diagrams. Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors).

Use Case diagram for the project is as follows:

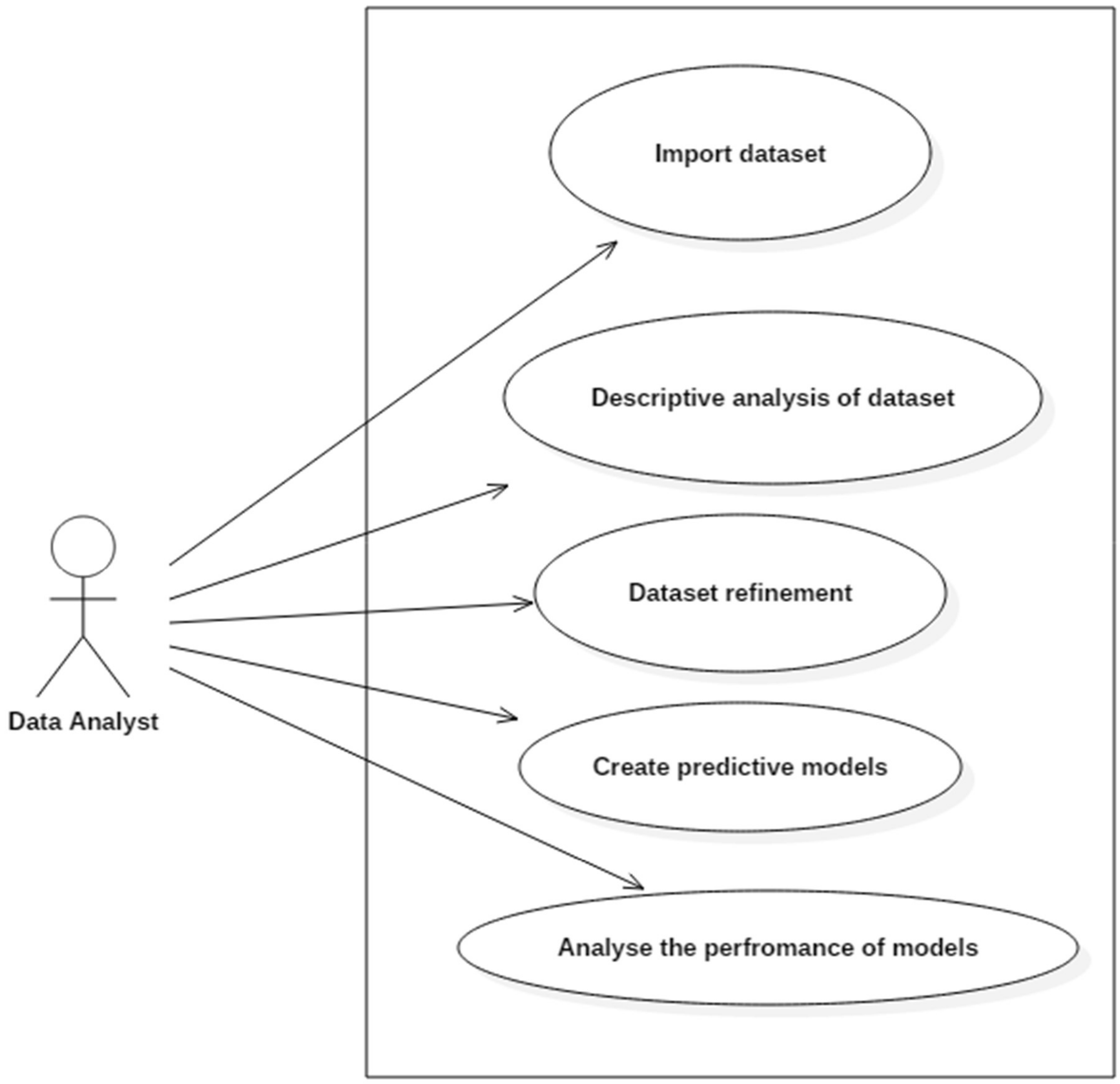

**Fig 4.4** Use case Diagram

## 4.6 Flow Chart Diagram

A **flowchart** is a type of diagram that represents an algorithm, workflow or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

Flow chart diagram for the project is as follows:

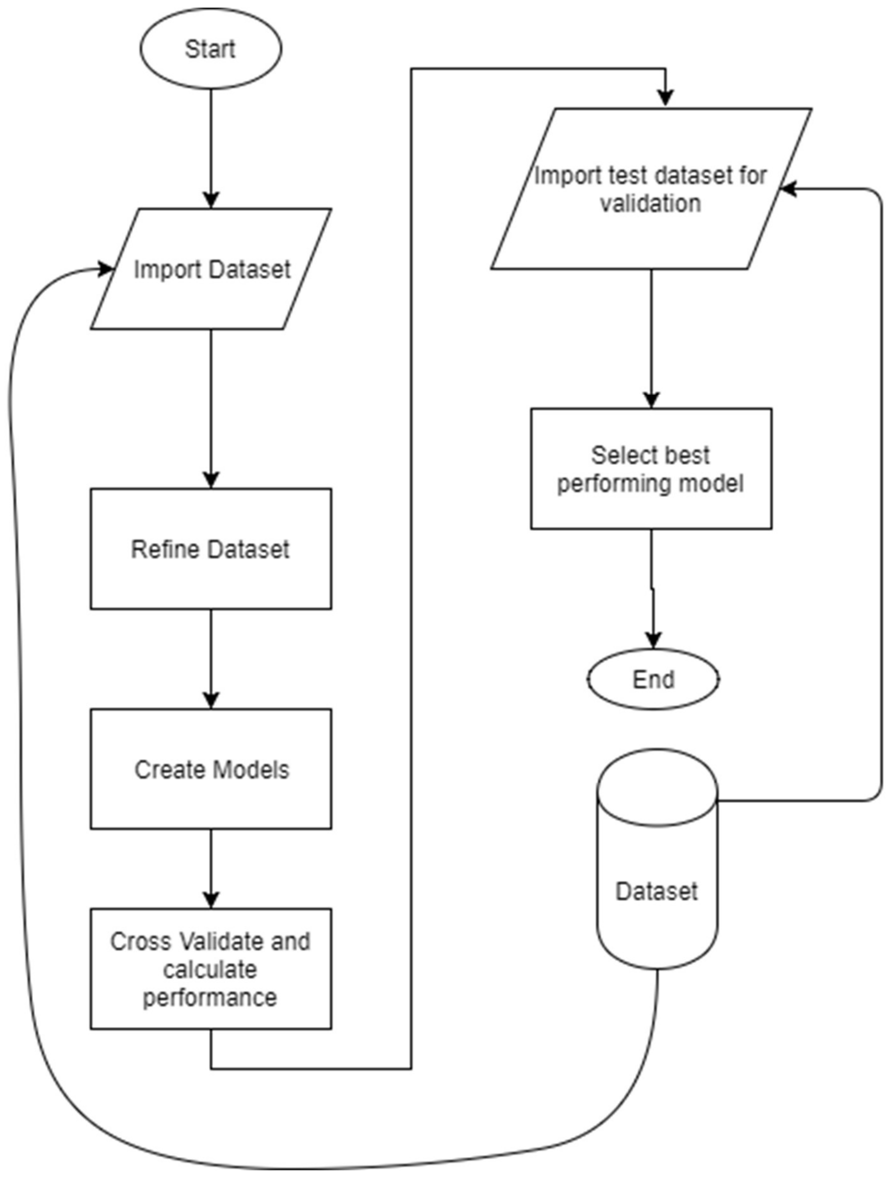
Start
Import Dataset
Refine Dataset
Create Models
Cross Validate and calculate performance
Import test dataset for validation
Select best performing model
End
Dataset

**Fig 4.5** Flow Chart Diagram

## 4.7 Activity Diagram

**Activity diagrams** are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e. workflows). Activity diagrams show the overall flow of control.

Activity diagram for the project is as follows:

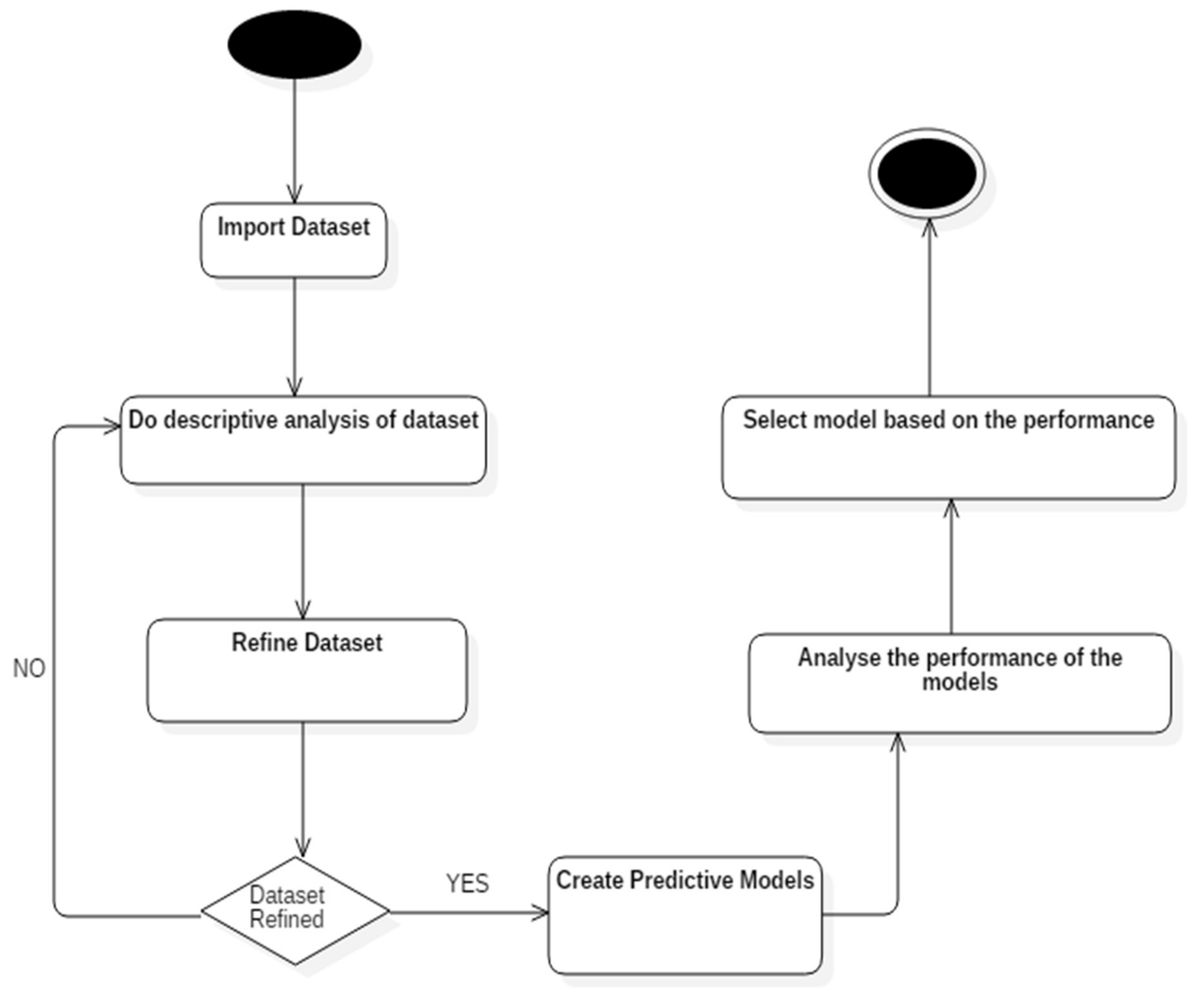
Import Dataset
Do descriptive analysis of dataset
Refine Dataset
NO
Dataset Refined
YES
Create Predictive Models
Analyse the performance of the models
Select model based on the performance

**Fig 4.6** Activity Diagram

## 4.8 Data Flow Diagram

A DFD shows what kind of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of process or information about whether processes will operate in sequence or in parallel (which is shown on a flowchart).

Data flow diagrams are also known as bubble charts. DFD is a designing tool used in the top-down approach to Systems Design. This context-level DFD is next "exploded", to produce a Level 1 DFD that shows some of the detail of the system being modelled. The Level 1 DFD shows how the system is divided into sub-systems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present in order for the system to do its job, and shows the flow of data between the various parts of the system.

Data flow diagrams are one of the three essential perspectives of the structured-systems analysis and design method SSADM. The sponsor of a project and the end users will need to be briefed and consulted throughout all stages of a system's evolution. With a data flow diagram, users are able to visualize how the system will operate, what the system will accomplish, and how the system will be implemented. The old system's dataflow diagrams can be drawn up and compared with the new system's data flow diagrams to draw comparisons to implement a more efficient system. Data flow diagrams can be used to provide the end user with a physical idea of where the data they input ultimately has an effect upon the structure of the whole system from order to dispatch to report. How any system is developed can be determined through a data flow diagram model.

In the course of developing a set of levelled data flow diagrams the analyst/designer is forced to address how the system may be decomposed into component sub-systems, and to identify the transaction data in the data model.

Data flow diagrams can be used in both Analysis and Design phase of the SDLC.

Data flow diagram for this project is given in the next page

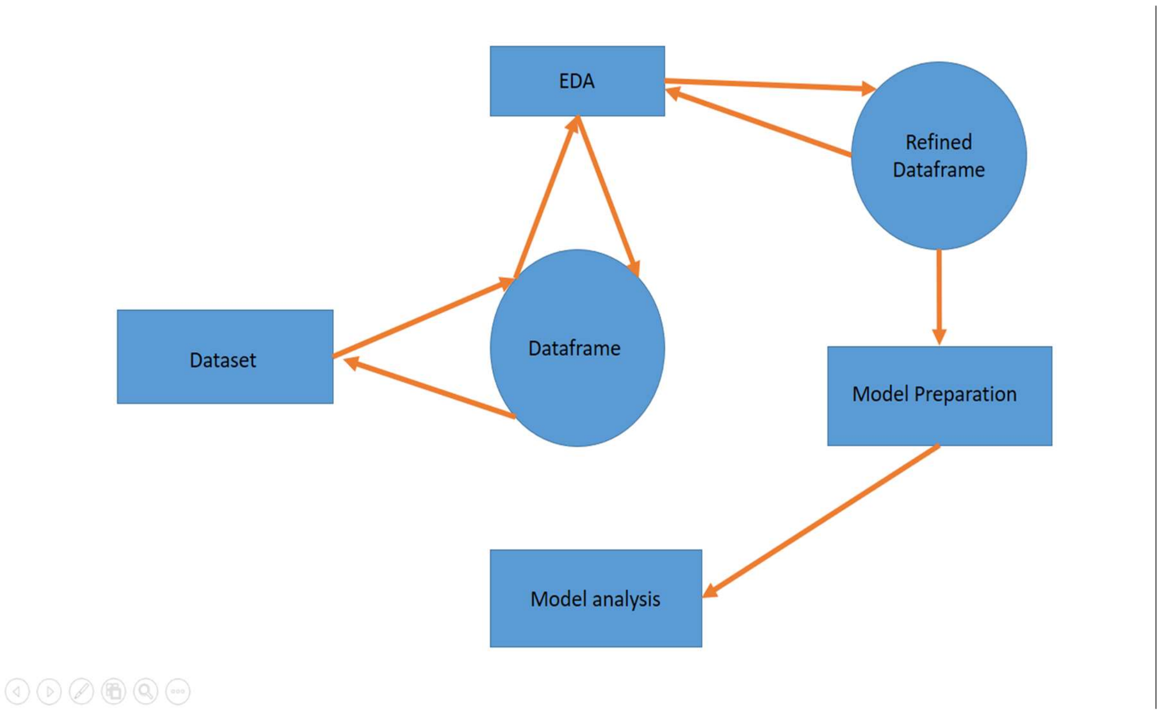

**Fig 4.7** Data Flow Diagram

## 4.9 Sequence Diagram

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

A sequence diagram shows, as parallel vertical lines (*lifelines*), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

Sequence Diagram for the project is as follows:

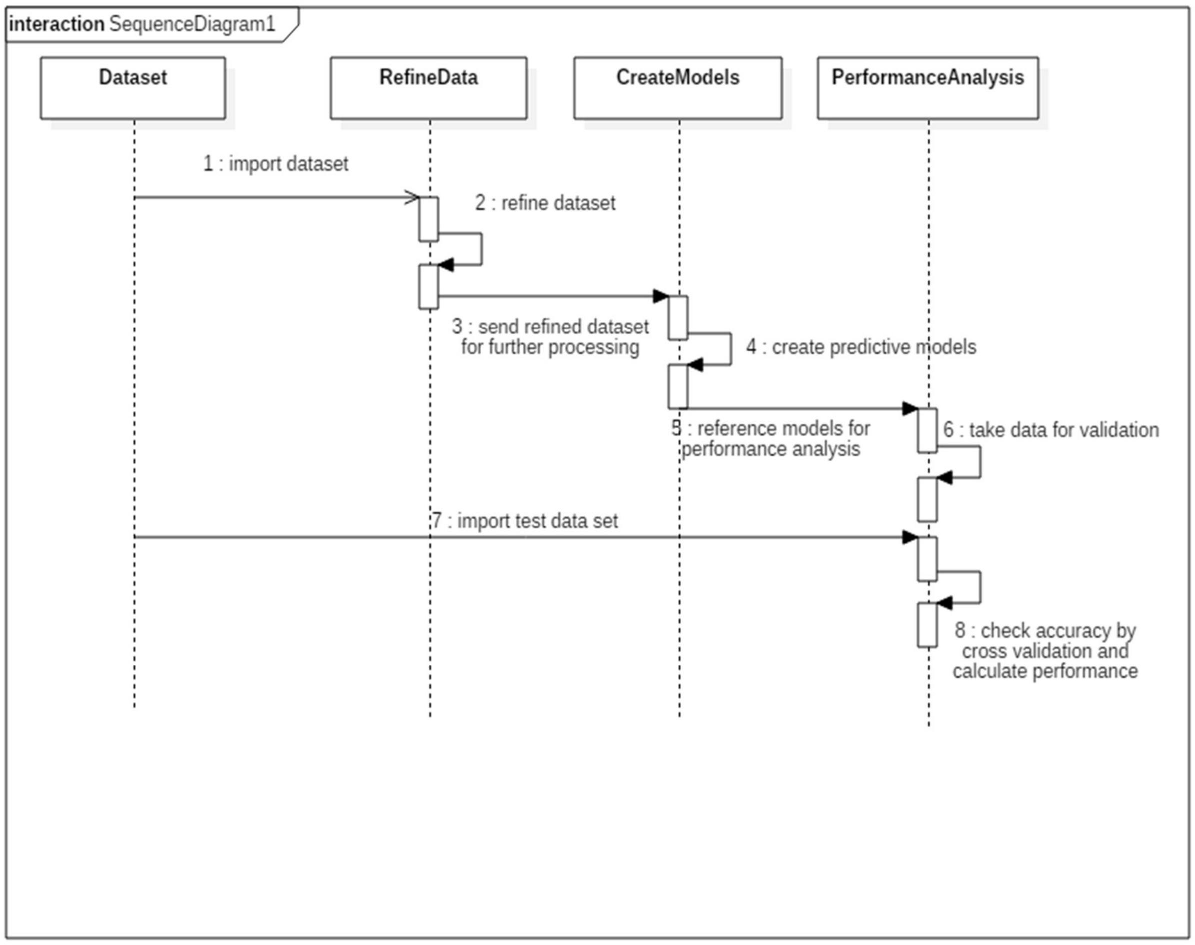
interaction SequenceDiagram1
Dataset
RefineData
CreateModels
PerformanceAnalysis
1 : import dataset
2 : refine dataset
3 : send refined dataset for further processing
4 : create predictive models
5 : reference models for performance analysis
6 : take data for validation
7 : import test data set
8 : check accuracy by cross validation and calculate performance

**Fig 4.8** Sequence Diagram

## 4.10 State Transition Diagram

State transition diagrams have been used right from the beginning in object-oriented modelling. The basic idea is to define a machine that has a number of states (hence the term finite state machine). The machine receives events from the outside world, and each event can cause the machine to transition from one state to another. For an example, take a look at figure 1. Here the machine is a bottle in a bottling plant. It begins in the empty state. In that state it can receive squirt events. If the squirt event causes the bottle to become full, then it transitions to the full state, otherwise it stays in the empty state (indicated by the transition back to its own state). When in the full state the cap event will cause it to transition to the sealed state.

The diagram indicates that a full bottle does not receive squirt events, and that an empty bottle does not receive cap events. Thus you can get a good sense of what events should occur, and what effect they can have on the object.

State transition diagram for the project is as follows:

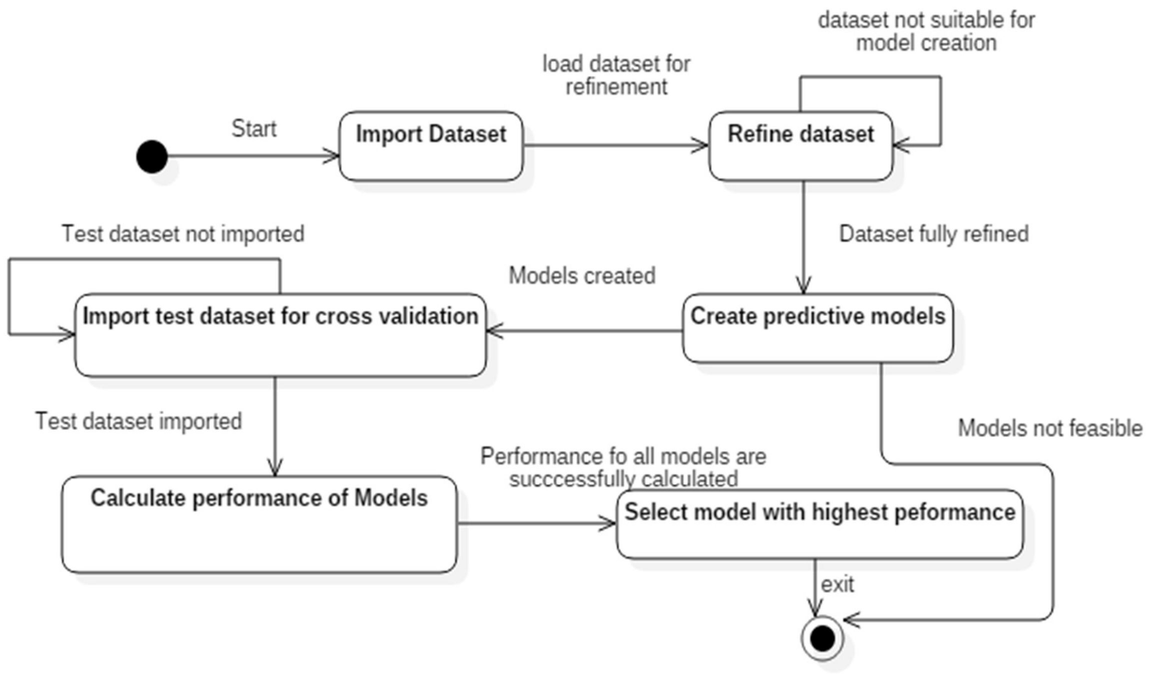
dataset not suitable for model creation
load dataset for refinement
Start
Import Dataset
Refine dataset
Test dataset not imported
Dataset fully refined
Models created
Import test dataset for cross validation
Create predictive models
Test dataset imported
Models not feasible
Performance fo all models are succcessfully calculated
Calculate performance of Models
Select model with highest peformance
exit

**Fig 4.9** State Transition Diagram

## 4.11 Deployment Diagram

Deployment diagrams are used to visualize the topology of the physical components of a system, where the software components are deployed.

Deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships.

The term Deployment itself describes the purpose of the diagram. Deployment diagrams are used for describing the hardware components, where software components are deployed. Component diagrams and deployment diagrams are closely related.

Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware.

UML is mainly designed to focus on the software artifacts of a system. However, these two diagrams are special diagrams used to focus on software and hardware components.

Most of the UML diagrams are used to handle logical components but deployment diagrams are made to focus on the hardware topology of a system. Deployment diagrams are used by the system engineers.

The purpose of deployment diagrams can be described as −

- Visualize the hardware topology of a system.
- Describe the hardware components used to deploy software components.
- Describe the runtime processing nodes.

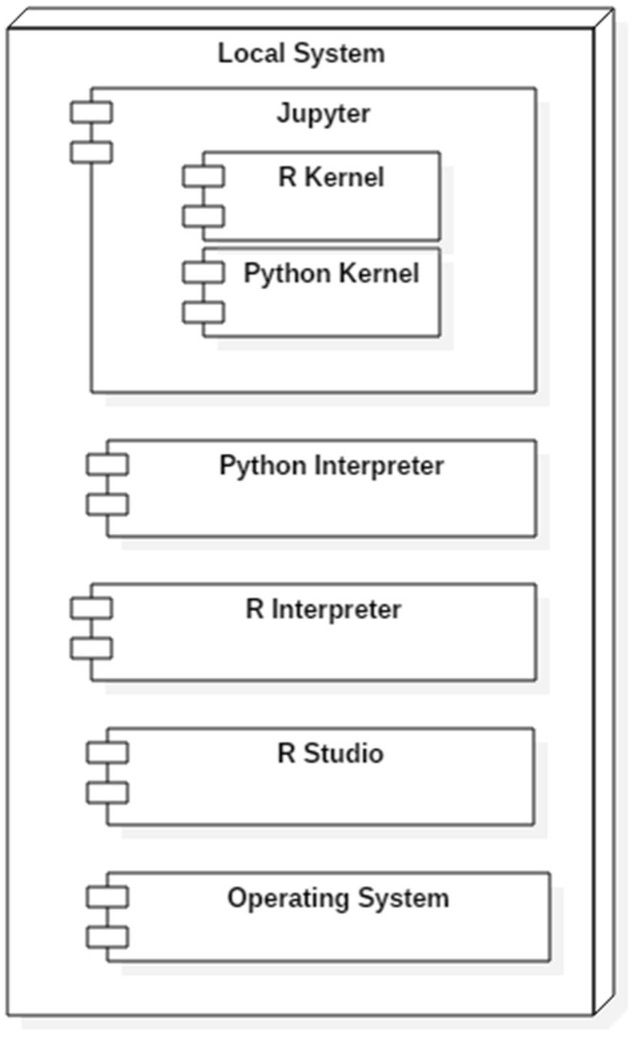

**Fig 4.10** Deployment Diagram

## 4.12 Collaboration Diagram

A collaboration diagram is a type of visual presentation that shows how various software objects interact with each other within an overall IT architecture and how users can benefit from this collaboration. A collaboration diagram often comes in the form of a visual chart that resembles a flow chart. It can show, at a glance, how a single piece of software complements other parts of a greater system

In many cases, a collaboration diagram will show how a system made up of individual software pieces works in real time. In other cases, the flow chart objects may represent a more abstract interaction, such as a general cause-and-effect or event-driven collaboration that may happen over time.

Generally, the labels on a collaboration diagram are determined by the needs of the user base. Someone creating this kind of resource may use actual file names, generic phrases representing the function of programs, or even customized icons to show how pieces of a system work together. Customized collaboration diagrams can help business leaders and others to see more about what’s going on within a complex IT system and how software interactions work.

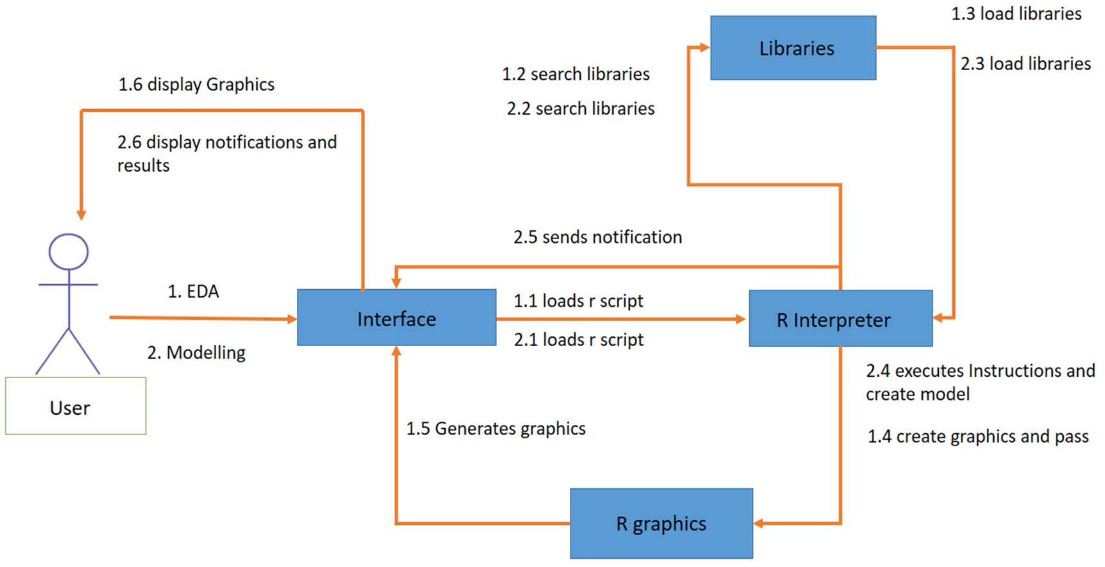

**Fig 4.11** Collaboration Diagram